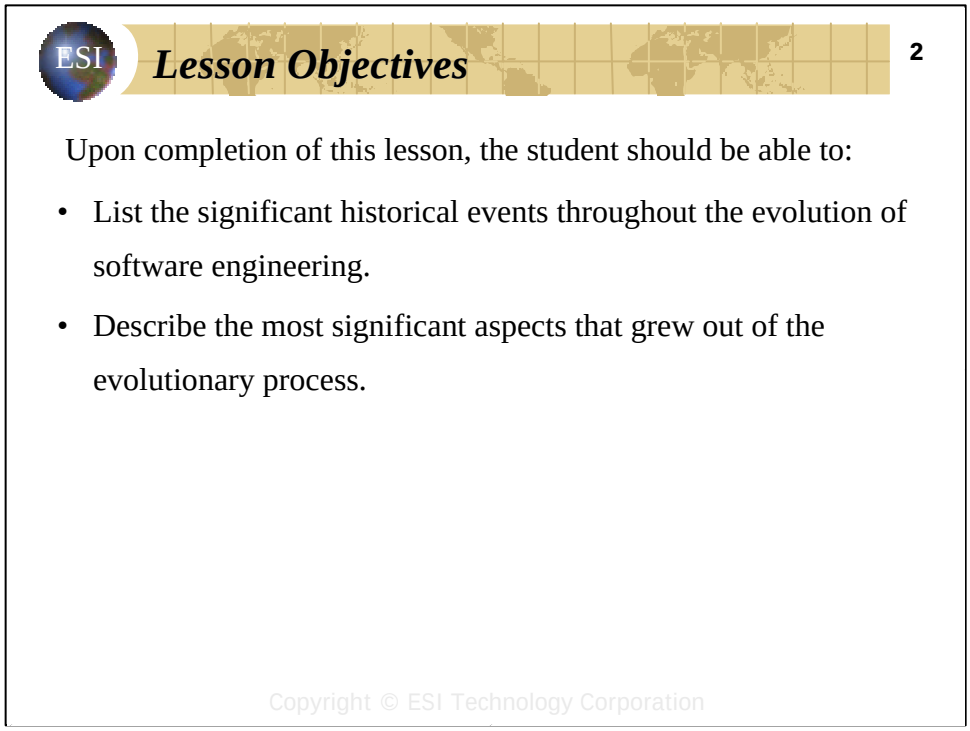
This lesson covers some significant historical events that lead to the object oriented paradigm.



Read and understand the objectives of this lesson. We will take a short tour of the most significant historical events that lead to present-day object orientation principles.

Software Evolution: Early1950's

- Assemblers introduced symbolic representation of the underlying machine instructions
- FORTRAN introduced:
  - variables
  - arrays
  - control structures

Problem: FORTRAN variable names would conflict in different parts of a program

Lets start by reviewing some historical events. Assemblers were the earliest languages that introduced symbolic representation of the underlying machine instructions. Although the programmer was still working at a very low level within the machine, interaction with the machine was at a symbolic level rather than at a numeric representation level. Symbolic instructions were generally aimed at memory locations that had symbolic names. For example: Add Cnt Total

FORTRAN was the first milestone of high-level programming; it introduced new concepts such as: Variables, Arrays and Control Structures. It was specialized in mathematical operations and flourished. In a general sense, it helped evolve software engineering languages to higher levels of abstraction.

In general, Fortran introduced the concept of procedural programming. Associated with this concept of programming were the problems that come along with a lack of data encapsulation. Data was generally manipulated in memory. Managing data in very large programs was a challenge.

Software Evolution: Late 1950's

- High-level programming languages such as PL/1, COBOL and Algol succeed FORTRAN
- Algol provides barriers to isolate variable names within program segments
  - Begin...End Blocks
  - Variables appearing within a block are scoped to that block
  - First attempt at providing protection or "encapsulation" within a programming language

In the late 50's, other high-level programming languages such as PL/1, COBOL and Algol evolved beyond FORTRAN. These languages were targeted for business applications whereas FORTRAN was used in scientific applications.

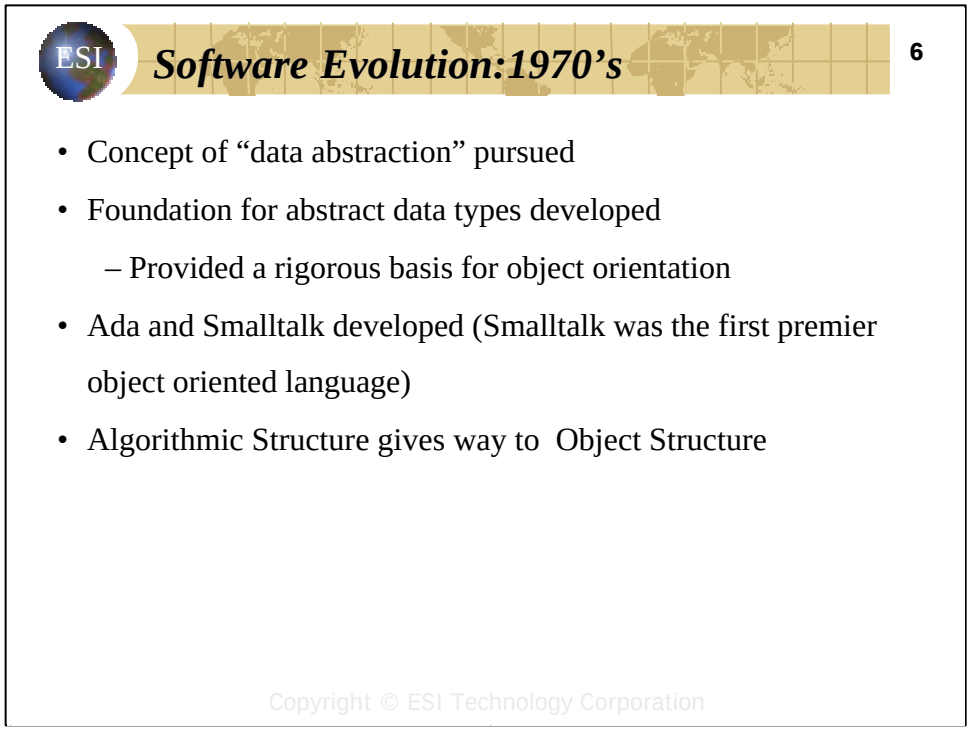
Algol was one of the first languages to provided barriers to separate variable names within program segments, hence the birth of Begin...End Blocks (ALGOL 60). Additionally, variable names appearing within a block appear only to that block, hence the concept of variable scoping was introduced. This was a first attempt at providing protection or "encapsulation" within a programming language.

Software Evolution:1960's

- Third Generation languages developed: Pascal, Simula, MUMPS, etc.
- Simula gave birth to the concept of "classes" and formed the basis for object orientation
- Start of structured programming
- Concept of "data abstraction" evolved

Within the 1960's, third generation languages like Pascal, Simula,... (and MUMPS!) were developed. In particular, Simula is considered to be the language that gave birth to Object-Orientation. The concept of "classes" was first introduced and formed the basis for today's concept of Object Orientation.

Structured Programming was introduced. The movement to data abstraction was underway.



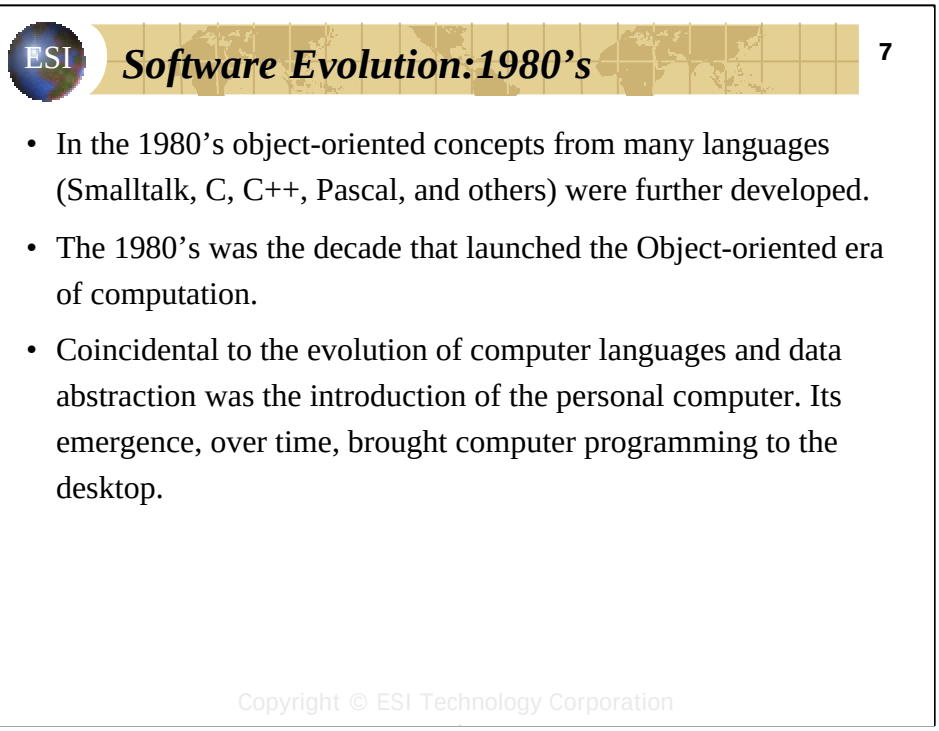
The 1970's evolved the art of computer programming to a new level. Within the United States, we had entered into the 'post Sputnik' era. That era produced large numbers of engineers and scientist. By the 1970's, the growing demand for programmers attracted a large number of these individuals. Educational institutions were starting to offer degrees in computer science.

Coincidentally, the concept of programming-in-the-large (calling modules/macros) evolved from programming-in-the-small (one large program). This move put pressure on the hardware, and software industry to evolve computer languages, databases, development tools to new levels.

Concept of "data abstraction" evolved even further - forming the foundation theory for abstract data types.

Programming languages Ada and Smalltalk were developed (Smalltalk was the first premier Object Oriented language).

Algorithmic and procedural structure started to gives way to object structure.



The 1980's launched the era of Object Orientation.

One can see that over the decades, software has evolved. As the needs of programmers and end users have become more complex and required more flexibility, programming languages have developed capabilities to provide for the increasing demands. Object Orientation has been called a *revolution* in software development. Yet it is more an *evolution* than a revolution.

With the emergence of the personal computer, computer programming was to take on a whole new significance.

Software Evolution:1990's

- In the 1990's, object orientation took off and established itself for the next generation of software engineering.
- C++ (hybrid of C) won the language wars over Smalltalk and other OO languages.
- Java, a refinement of C++ crashed upon the scene and established itself as a compromise be between C++ and environment oriented languages like Smalltalk.
- Object Oriented databases made their debut and are in the process of finding their spot in the world of Information Technology.

With the advent of the personal computer and distributed computing, object orientation took on a whole new meaning. By the 1990's, it had evolved into a combined set of features that we based on good software engineering principles. (These principles will be developed in the next lesson.)

Competition produced an apparent winner in the object oriented programming language wars. C++ appeared to have the most support. It implemented good, sound OO features and provided excellent performance, However, it was not without its faults.

As an answer to many of the problems that come along with C++ (mainly memory leaks and hard to use), Sun introduced Java. Java evolved the C++ language into a simpler-to-use language that resides in an environment and avoids the memory management problems of C++. It married many of the best features of C++ and Smalltalk - another point on the evolutionary path to object oriented programming languages.

Within the 90's, Object Oriented Database Management Systems (OODBMS) were introduced. In general, it was assumed that the IT industry would readily leave relational technology behind to reap the benefits of object orientation. This did not happen because of numerous reasons. First, object orientation requires a major leap in knowledge - old knowledge gets in the way. Organizations that boldly tried OODBMS implementation often created object models that were reminiscent of 'old design' criteria. We simply did not have the rich OO knowledge base that we have today (Check out the number of books on OO design patterns). Additionally, most of the OODBMS implementations were new and immature. We can go on to list other reasons for the false start, but suffice it to say, OODBMS systems are finding their niche. The Internet is a natural place for this technology.

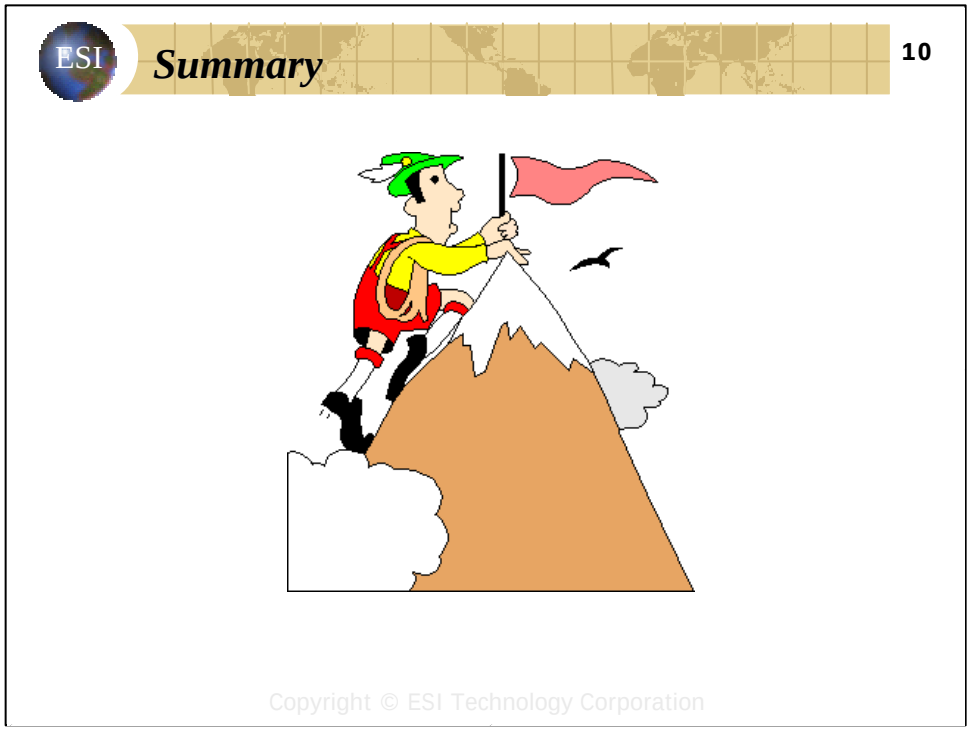
Within the 1990's, attempts were made to evolve the MUMPS database and language technology to objects. ESI Technology Corporation lead the effort followed by InterSystems Corporation. These two implementations will be used throughout this tutorial.

Software Evolution: The Next Decade?

- Over the decades, as needs have become more complex and required more flexibility, languages have evolved with new capabilities.
- Today, businesses are putting more demands on technology, particularly the move to the Internet.
- Object orientation has been called a *revolution* in software development. Yet it is more an *evolution* than a revolution.

Where will technology lead software engineering within the next decade? We have already seen a strong rally around object orientation. Not only have strong object oriented languages evolved over the last two decades, OODBMS systems have evolved. Languages like Java that have targeted the Internet, entrench object orientation even more as the ‘paradigm of choice’.

Although this was a short tour through the history of computing (sufficient for the tutorial), the object oriented paradigm evolved to where it is today - a powerful concept that models the world as it exists. More importantly, because the object paradigm provides the features needed to build abstract or generalized solutions, it provides for extensibility not realized before. Additionally, it offers accelerated development due to reusability.

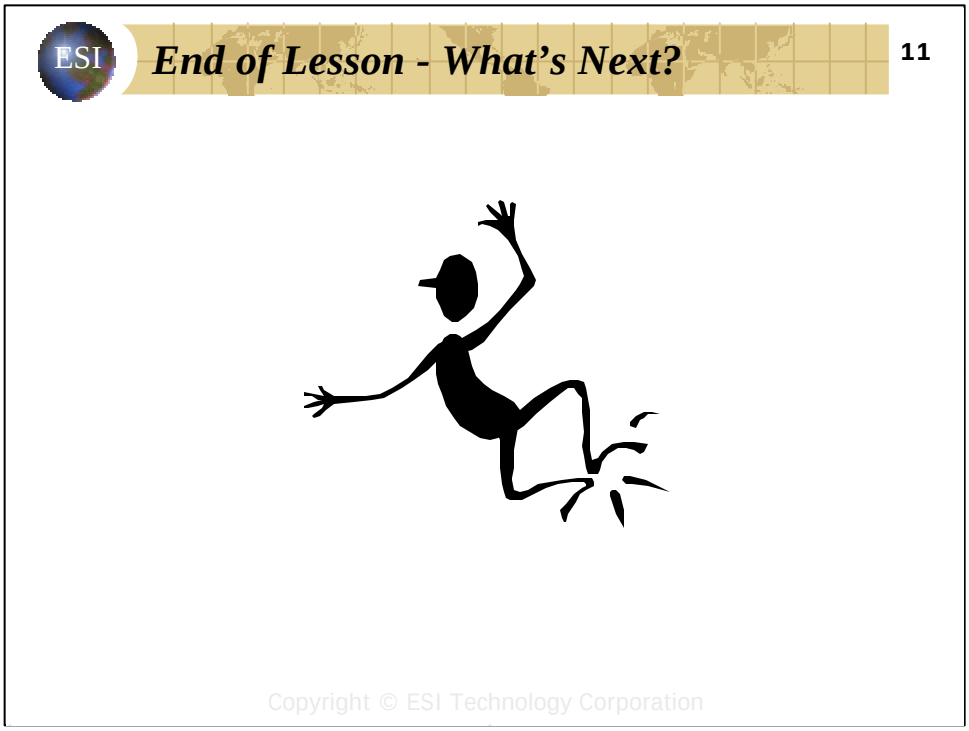


What have we inherited from the short history of software engineering that has lead to present day object oriented concepts?

First and foremost is the concept of **abstraction**. This evolved out of a need to generalize software modules and to provide reusable components. One OO feature that evolved out of this is the concepts of classification and inheritance - the ability to derive definitional information and behavior from ancestors.

Another fundamental concept that evolved is the concept of **encapsulation**, or **information hiding**. It became important for programs to protect data. The initial efforts, as we saw, started with variable scoping. Today, the concept of encapsulation is absolutely fundamental to good software quality.

Although not explicitly talked about in our historical journey, the concept of **loose binding** became more and more important to extensibility. Tightly bound software packages were not reusable - forcing wholesale rewrites. Distributed systems are dependent upon loosely bound software packages. Inherent to loose binding are well defined interfaces and a messaging.



The next lesson, Lesson 2, will concentrate on laying a rigorous foundation for Object Orientation as explained in Bertrand Meyer's book **Object-Oriented Software Construction.** The principles Meyer derives, although not used directly by engineers, should always provide guidance to developing quality software packages.

Proceed to the next lesson titled A Foundation for Object Orientation when it is published.